**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Операционные Системы»

**Лабораторная работа № 6**

Тема: Управлении серверами сообщений, применение отложенных вычислений, интеграция программных систем друг с другом

Студент: Семин Александр Витальевич

Группа: М8О-206Б-20

Преподаватель: Соколов Андрей Алексеевич

Дата:

Оценка:

Москва, 2021

1. **Постановка задачи**

Реализовать распределенную систему по асинхронной обработке запросов. В данной распределенной системе должно существовать 2 вида узлов: «управляющий» и «вычислительный». Необходимо объединить данные узлы в соответствии с той топологией, которая определена вариантом. Связь между узлами необходимо осуществить при помощи

технологии очередей сообщений. Также в данной системе необходимо предусмотреть проверку доступности узлов в соответствии с вариантом. При убийстве («kill -9») любого вычислительного узла система должна пытаться максимально сохранять свою работоспособность, а именно все

дочерние узлы убитого узла могут стать недоступными, но родительские узлы должны сохранить свою работоспособность.

Управляющий узел отвечает за ввод команд от пользователя и отправку этих команд на вычислительные узлы.

Вариант 39:

**Топология** - узлы находятся в идеально сбалансированном бинарном дереве. Каждый следующий узел должен добавляться в самое наименьшее поддерево.

**Набор команд** – поиск подстроки в строке

**Проверка доступности узла**: ping id.

1. **Набор тестов**

*Первый тестовый набор:*

progger@asus:~/Desktop/OS\_labs/l6$ ./server

Menu:

1. create <ID> -- create a node

2. exec <ID> <STR1> <STR2> -- search STR2 in STR1

3. ping <ID> -- check node existence

3. kill <ID> -- delete a calculation node

Please enter your command:> create 5

Ok: 7746

> create 2

Ok: 7771

> create 7

Ok: 7774

> ping 2

Ok: 1

> kill 2

Ok

> ping 2

Ok: 0

> exec 7 3 1 2 3

Ok: 7: 6

1. **Листинг программы**

**ZMQFunctions.h**

#pragma once

#include <bits/stdc++.h>

#include <zmq.hpp>

const int MAIN\_PORT = 4040;

void send\_message(zmq::socket\_t &socket, const std::string &msg) {

zmq::message\_t message(msg.size());

memcpy(message.data(), msg.c\_str(), msg.size());

socket.send(message);

}

std::string receive\_message(zmq::socket\_t &socket) {

zmq::message\_t message;

int chars\_read;

try {

chars\_read = (int)socket.recv(&message);

}

catch (...) {

chars\_read = 0;

}

if (chars\_read == 0) {

throw -1;

}

std::string received\_msg(static\_cast<char\*>(message.data()), message.size());

return received\_msg;

}

void connect(zmq::socket\_t &socket, int port) {

std::string address = "tcp://127.0.0.1:" + std::to\_string(port);

socket.connect(address);

}

void disconnect(zmq::socket\_t &socket, int port) {

std::string address = "tcp://127.0.0.1:" + std::to\_string(port);

socket.disconnect(address);

}

int bind(zmq::socket\_t &socket, int id) {

int port = MAIN\_PORT + id;

std::string address = "tcp://127.0.0.1:" + std::to\_string(port);

while(1){

try{

socket.bind(address);

break;

}

catch(...){

port++;

}

}

return port;

}

void unbind(zmq::socket\_t &socket, int port) {

std::string address = "tcp://127.0.0.1:" + std::to\_string(port);

socket.unbind(address);

}

**BalancedTree.h**

#ifndef BALANCED\_TREE\_H

#define BALANCED\_TREE\_H

#include <bits/stdc++.h>

class BalancedTree {

class BalancedTreeNode {

public:

int id;

BalancedTreeNode\* left;

BalancedTreeNode\* right;

int height;

bool available;

BalancedTreeNode (int id) {

this->id = id;

available = true;

left = NULL;

right = NULL;

}

void CheckAvailability (int id) {

if (this->id == id){

available = false;

}

else {

if (left != NULL) {

left->CheckAvailability(id);

}

if (right != NULL) {

right->CheckAvailability(id);

}

}

}

void Remove (int id, std::set<int> &ids) {

if (left != NULL && left->id == id) {

left->RecursionRemove(ids);

ids.erase(left->id);

delete left;

left = NULL;

}

else if (right != NULL && right->id == id) {

right->RecursionRemove(ids);

ids.erase(right->id);

delete right;

right = NULL;

}

else {

if (left != NULL) {

left->Remove(id, ids);

}

if (right != NULL) {

right->Remove(id, ids);

}

}

}

void RecursionRemove (std::set<int> &ids) {

if (left != NULL) {

left->RecursionRemove(ids);

ids.erase(left->id);

delete left;

left = NULL;

}

if (right != NULL) {

right->RecursionRemove(ids);

ids.erase(right->id);

delete right;

right = NULL;

}

}

void AddInNode (int id, int parent\_id, std::set<int> &ids) {

if (this->id == parent\_id) {

if (left == NULL){

left = new BalancedTreeNode(id);

}

else {

right = new BalancedTreeNode(id);

}

ids.insert(id);

}

else {

if (left != NULL) {

left->AddInNode(id, parent\_id, ids);

}

if (right != nullptr) {

right->AddInNode(id, parent\_id, ids);

}

}

}

int MinimalHeight() {

if (left == NULL || right == NULL) {

return 0;

}

int left\_height = -1;

int right\_height = -1;

if (left != NULL && left->available == true) {

left\_height = left->MinimalHeight();

}

if (right != NULL && right->available == true) {

right\_height = right->MinimalHeight();

}

if (right\_height == -1 && left\_height == -1) {

available = false;

return -1;

}

else if (right\_height == -1) {

return left\_height + 1;

}

else if (left\_height == -1) {

return right\_height + 1;

}

else {

return std::min(left\_height, right\_height) + 1;

}

}

int IDMinimalHeight(int height, int current\_height) {

if (height < current\_height) {

return -2;

}

else if (height > current\_height) {

int current\_id = -2;

if (left != NULL && left->available == true) {

current\_id = left->IDMinimalHeight(height, (current\_height + 1));

}

if (right != NULL && right->available == true && current\_id == -2){

current\_id = right->IDMinimalHeight(height, (current\_height + 1));

}

return current\_id;

}

else {

if (left == NULL || right == NULL){

return id;

}

return -2;

}

}

~BalancedTreeNode() {}

};

private:

BalancedTreeNode\* root;

public:

std::set<int> ids;

BalancedTree() {

root = new BalancedTreeNode(-1);

}

bool Exist(int id) {

if (ids.find(id) != ids.end()) {

return true;

}

return false;

}

void AvailabilityCheck(int id) {

root->CheckAvailability(id);

}

int FindID() {

int h = root->MinimalHeight();

return root->IDMinimalHeight(h, 0);

}

void AddInTree(int id, int parent) {

root->AddInNode(id, parent, ids);

}

void RemoveFromRoot(int idElem) {

root->Remove(idElem, ids);

}

~BalancedTree() {

root->RecursionRemove(ids);

delete root;

}

};

#endif

**CalculationNode.h**

#include <bits/stdc++.h>

#include "ZMQFunctions.h"

#include "unistd.h"

class CalculationNode {

private:

zmq:: context\_t context;

public:

zmq:: socket\_t left, right, parent;

int id, left\_id = -2, right\_id = -2, parent\_id;

int left\_port, right\_port, parent\_port;

CalculationNode(int id, int parent\_port, int parent\_id):

id(id),

parent\_port(parent\_port),

parent\_id(parent\_id),

left(context, ZMQ\_REQ),

right(context, ZMQ\_REQ),

parent(context, ZMQ\_REP)

{

if (id != -1) {

connect(parent, parent\_port);

}

}

std:: string create (int child\_id) {

int port;

bool isleft = false;

if (left\_id == -2) {

left\_port = bind(left, child\_id);

left\_id = child\_id;

port = left\_port;

isleft = true;

}

else if (right\_id == -2) {

right\_port = bind(right, child\_id);

right\_id = child\_id;

port = right\_port;

}

else {

std:: string fail = "Error: can not create the calculation node";

return fail;

}

int fork\_id = fork();

if (fork\_id == 0) {

if (execl("./client", "client", std:: to\_string(child\_id).c\_str(), std:: to\_string(port).c\_str(), std:: to\_string(id).c\_str(), (char\*)NULL) == -1) {

std:: cout << "Error: can not run the execl-command" << std:: endl;

exit(EXIT\_FAILURE);

}

}

else {

std:: string child\_pid;

try {

if (isleft) {

left.setsockopt(ZMQ\_SNDTIMEO, 3000);

send\_message(left, "pid");

child\_pid = receive\_message(left);

}

else {

right.setsockopt(ZMQ\_SNDTIMEO, 3000);

send\_message(right, "pid");

child\_pid = receive\_message(right);

}

return "Ok: " + child\_pid;

}

catch (int) {

std:: string fail = "Error: can not connect to the child";

return fail;

}

}

}

std:: string ping (int id) {

std:: string answer = "Ok: 0";

if (this->id == id) {

answer = "Ok: 1";

return answer;

}

else if (left\_id == id) {

std:: string message = "ping " + std:: to\_string(id);

send\_message(left, message);

try {

message = receive\_message(left);

if (message == "Ok: 1") {

answer = message;

}

}

catch(int){}

}

else if (right\_id == id) {

std:: string message = "ping " + std:: to\_string(id);

send\_message(right, message);

try {

message = receive\_message(right);

if (message == "Ok: 1") {

answer = message;

}

}

catch(int){}

}

return answer;

}

std:: string sendstring (std:: string string, int id) {

std:: string answer = "Error: Parent not found";

if (left\_id == -2 && right\_id == -2) {

return answer;

}

else if (left\_id == id) {

if (ping(left\_id) == "Ok: 1") {

send\_message(left, string);

try{

answer = receive\_message(left);

}

catch(int){}

}

}

else if (right\_id == id) {

if (ping(right\_id) == "Ok: 1") {

send\_message(right, string);

try {

answer = receive\_message(right);

}

catch(int){}

}

}

else {

if (ping(left\_id) == "Ok: 1") {

std:: string message = "send " + std:: to\_string(id) + " " + string;

send\_message(left, message);

try {

message = receive\_message(left);

}

catch(int) {

message = "Error: Parent not found";

}

if (message != "Error: Parent not found") {

answer = message;

}

}

if (ping(right\_id) == "Ok: 1") {

std:: string message = "send " + std:: to\_string(id) + " " + string;

send\_message(right, message);

try {

message = receive\_message(right);

}

catch(int) {

message = "Error: Parent not found";

}

if (message != "Error: Parent not found") {

answer = message;

}

}

}

return answer;

}

std:: string exec (std:: string string) {

std:: istringstream string\_thread(string);

std::string s1, s2;

string\_thread >> s1;

string\_thread >> s2;

int res;

res = s1.find(s2);

std:: string answer = "Ok: " + std:: to\_string(id) + ": " + std:: to\_string(res);

return answer;

}

std:: string treeclear (int child) {

if (left\_id == child) {

left\_id = -2;

unbind(left, left\_port);

}

else {

right\_id = -2;

unbind(right, right\_port);

}

return "Ok";

}

std:: string kill () {

if (left\_id != -2){

if (ping(left\_id) == "Ok: 1") {

std:: string message = "kill";

send\_message(left, message);

try {

message = receive\_message(left);

}

catch(int){}

unbind(left, left\_port);

left.close();

}

}

if (right\_id != -2) {

if (ping(right\_id) == "Ok: 1") {

std:: string message = "kill";

send\_message(right, message);

try {

message = receive\_message(right);

}

catch (int){}

unbind(right, right\_port);

right.close();

}

}

return std:: to\_string(parent\_id);

}

~CalculationNode() {}

};

**Server.cpp**

#include <bits/stdc++.h>

#include "CalculationNode.h"

#include "ZMQFunctions.h"

#include "BalancedTree.h"

int main() {

std:: string command;

CalculationNode node(-1, -1, -1);

std:: string answer;

std:: cout << "Menu:\n"

"\t1. create <ID> -- create a node\n"

"\t2. exec <ID> <STR1> <STR2> -- search STR2 in STR1\n"

"\t3. ping <ID> -- check node existence\n"

"\t3. kill <ID> -- delete a calculation node\n";

BalancedTree tree;

while ((std:: cout << "Please enter your command:\n") && (std:: cin >> command)) {

if (command == "create") {

int child;

std:: cin >> child;

if (tree.Exist(child)) {

std:: cout << "Error: Already exists" << std:: endl;

}

else {

while (true) {

int idParent = tree.FindID();

if (idParent == node.id) {

answer = node.create(child);

tree.AddInTree(child, idParent);

break;

}

else {

std:: string message = "create " + std:: to\_string(child);

answer = node.sendstring(message, idParent);

if (answer == "Error: Parent not found") {

tree.AvailabilityCheck(idParent);

}

else {

tree.AddInTree(child, idParent);

break;

}

}

}

std:: cout << answer << std::endl;

}

}

else if (command == "exec") {

std:: string str;

int child;

std:: cin >> child;

getline(std:: cin, str);

if (!tree.Exist(child)) {

std:: cout << "Node doesn't exist\n";

}

else {

std:: string message = "exec " + str;

answer = node.sendstring(message, child);

std:: cout << answer << std:: endl;

}

}

else if (command == "ping") {

int child;

std:: cin >> child;

if (!tree.Exist(child)) {

std::cout << "Ok: 0" << std:: endl;

}

else if (node.left\_id == child || node.right\_id == child) {

answer = node.ping(child);

std:: cout << answer << std:: endl;

}

else {

std:: string message = "ping " + std:: to\_string(child);

answer = node.sendstring(message, child);

if (answer == "Error: Parent not found") {

answer = "Ok: 0";

}

std:: cout << answer << std:: endl;

}

}

else if (command == "kill") {

int child;

std:: cin >> child;

std:: string message = "kill";

if (!tree.Exist(child)) {

std:: cout << "Error: Parent is not existed" << std:: endl;

}

else {

answer = node.sendstring(message, child);

if (answer != "Error: Parent not found") {

tree.RemoveFromRoot(child);

if (child == node.left\_id){

unbind(node.left, node.left\_port);

node.left\_id = -2;

answer = "Ok";

}

else if (child == node.right\_id) {

node.right\_id = -2;

unbind(node.right, node.right\_port);

answer = "Ok";

}

else {

message = "clear " + std:: to\_string(child);

answer = node.sendstring(message, std:: stoi(answer));

}

std:: cout << answer << std:: endl;

}

}

}

else {

std:: cout << "Please enter correct command!" << std:: endl;

}

}

node.kill();

return 0;

}

**Client.cpp:**

#include <bits/stdc++.h>

#include "CalculationNode.h"

#include "ZMQFunctions.h"

#include "BalancedTree.h"

int main(int argc, char \*argv[]) {

if (argc != 4) {

std:: cout << "Usage: 1)./main, 2) child\_id, 3) parent\_port, 4) parent\_id" << std:: endl;

exit(EXIT\_FAILURE);

}

CalculationNode node(atoi(argv[1]), atoi(argv[2]), atoi(argv[3]));

while(true) {

std:: string message;

std:: string command;

message = receive\_message(node.parent);

std:: istringstream request(message);

request >> command;

if (command == "pid") {

std:: string answer = std:: to\_string(getpid());

send\_message(node.parent, answer);

}

else if (command == "ping") {

int child;

request >> child;

std:: string answer = node.ping(child);

send\_message(node.parent, answer);

}

else if (command == "create") {

int child;

request >> child;

std:: string answer = node.create(child);

send\_message(node.parent, answer);

}

else if (command == "exec") {

std::string str;

getline(request, str);

std::string answer = node.exec(str);

send\_message(node.parent, answer);

}

else if (command == "kill") {

std:: string answer = node.kill();

send\_message(node.parent, answer);

disconnect(node.parent, node.parent\_port);

node.parent.close();

break;

}

}

return 0;

}

**Makefile**:

files: server client

server: Server.cpp

g++ -fsanitize=address Server.cpp -lzmq -o server -w

client: Client.cpp

g++ -fsanitize=address Client.cpp -lzmq -o client -w

clean:

rm -rf server client

1. **Выводы**

В процессе выполнения работы я реализовал распределенную систему по асинхронной обработке запросов. В программе используется протокол передачи данных tcp, в котором общение между процессами происходит через определенные порты.

Обмен сообщений происходит посредством функций библиотеки ZMQ, а в частности, ее паттерном «Request – Reply». Это один из самых простых и прямолинейных паттернов, который своей реализацией очень напоминает pipe. Также я в процессе выполнения работы освежил знания по реализации идеально сбалансированного бинарного дерева на С++.